CS3025 Compiladores

Laboratorio 8.3 – 6 octubre 2023

Analisis Semantico – Type Checking I

El folder lab8.3 contiene la solución del laboratorio 8.2, mas algunas modificaciones y adiciones para este laboratorio.
Estos programas implementan el parser, printer e intérprete de un lenguaje imperativo con declaraciones a diferentes niveles. El lenguaje tiene la siguiente sintaxis:

```
Program ::= Body
Body ::= VarDecList StatementList
VarDecList ::= (VarDec)*
VarDec ::= "var" Type VarList ";"
Type ::= id
VarList :: id ("," id)*
StatementList ::= Stm (";" Stm)* …
Stm ::=   id "=" CExp |
          "print" "(" CExp ")" |
          "if" CExp "then" Body ["else" Body] "endif" |
          "while" CExp "do" Body "endwhile"
```

Compilar y probar el programa con los ejemplos `ejemplo11.imp`, `ejemplo22.imp` y `ejemplo33.cpp`, sobre todo, `ejemplo222cpp`, funcionan.

En este laboratorio implementaremos un type checker para IMP. Esto involucrara modificar la definición de la tabla de símbolos y crear una interface y un visitor específico para este fin.

1.0 Modificando el Enviroment (Table de simbolos)

El intérprete de IMP utiliza un objeto de clase `Environment` para mapear variables a valores enteros. El type checker necesitara la misma funcionalidad e interface para mapear variables a strings (representaremos tipos con cadenas). Es un caso clásico del uso de templates.

La declaración y definición de `Environment` esta en un solo archivo, `environment.hh`. Modificar la clase para que pueda mapear variables a elementos de tipo T. Esto puede se hacer agregando

```
template <typename T>
class Environment {
```

a la definición de clase y reemplazando `int` con `T` en los lugares indicados.

Tambien necesitaremos actualizar `imp_interpreter.hh` con la inicializacion del template correcto. Compilar y ejecutar el nuevo programa.

2.0 Un visitor para Type Checking: `ImpTypeChecker`.

El typecheker será implementado con un visitor. Este visitor necesitara una interfaz nueva porque en lugar de retornar enteros necesitaremos retornar tipos, es decir, strings.

La nueva interface está definida en `type_visitor.hh`. Ademas, se han hecho los cambios necesarios a `imp.hh` e `imp.cpp` para que acepte el nuevo visitor. Tambien tenemos una implementación parcial del typechecker en `imp_typecker.cpp`. Nos hace falta el header file!

Hacer una copia de `imp_interpreter.hh` y llamarla imp_typechecker.hh. Hacer los cambios necesarios al nombre de clase y la clase de la cual hereda funcionalidad (visitor). Notar que para hacer type checking necesitamos un environment pero con mappings a otro tipo.

Modificar `test_imp_dec.ccp` (quitar comentarios) para que use el type checker. Compilar y ejecutar.

3.0 El Type Checker

Modificar `imp_typechecker.cpp` para implementación la verificación de tipos y otras verificaciones semánticas.

La verificación de tipo de expresiones implica verificar que los tipos de las subexpresiones (operación binaria) son los correctos. Además se deberá retornar el tipo de la expresión.
La verificación de tipos de sentencias no retorna ningún tipo pero si verifica que los tipos de las subexpresiones son los correctos e.g. las condiciones.

Esto lo haremos pasa a paso.